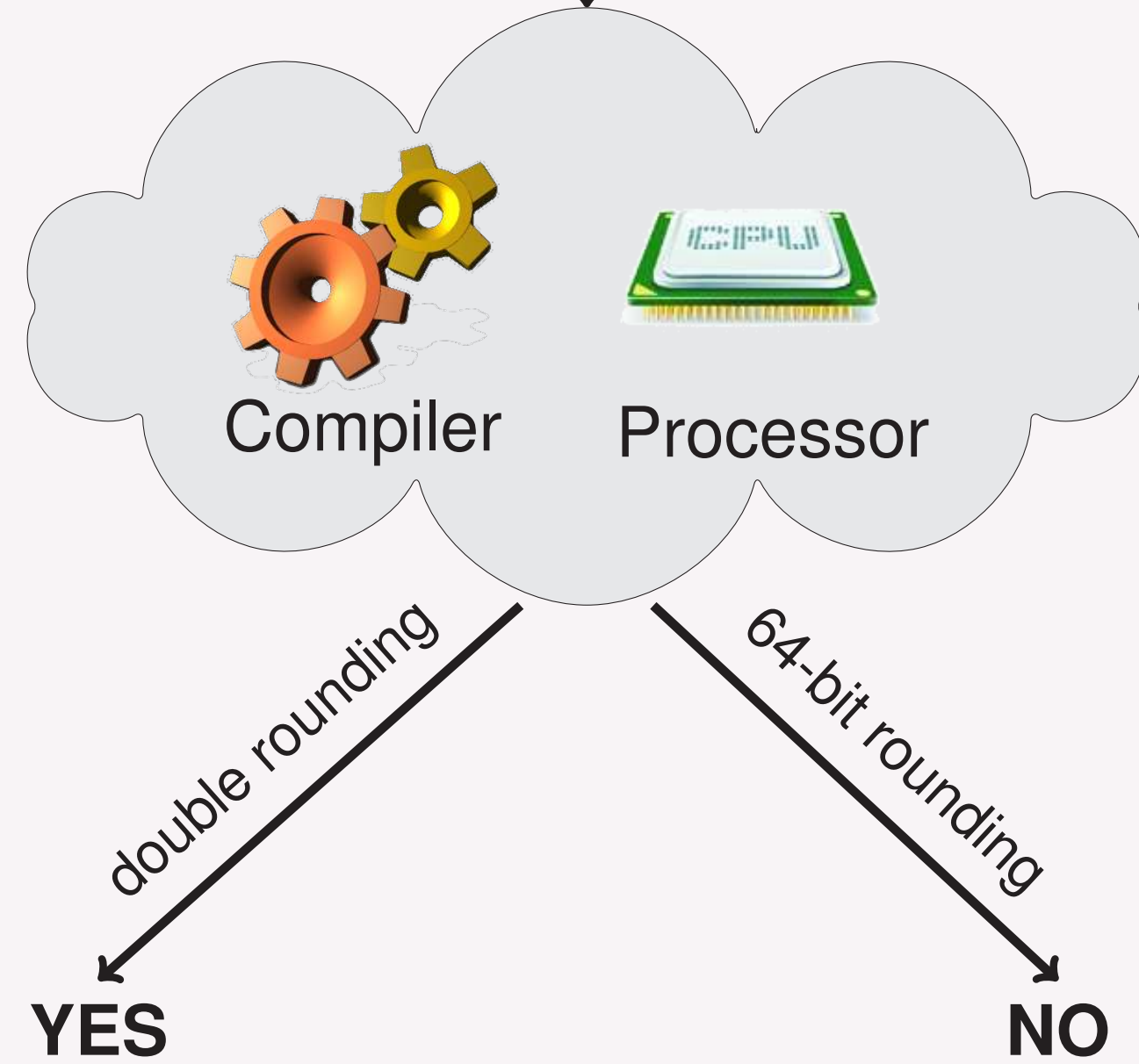


# FORMAL PROOFS OF NUMERICAL PROGRAMS

```
int main(){
  double x = 1.0;
  double y = 0x1p-53 + 0x1p-64;
  double z = x + y;
  if(z <= 1)
    printf("YES");
  else
    printf("NO");
}
```

--:--- double\_rounding.c Top (5,13)



How to prove a program whatever the compiler and the architecture?

Previous works: only follow the strict IEEE-754

State the rounding error of each FP computation whatever the environment

ACSL specification language

Annotated C program

Frama-C/Jessie plug-in

WHY verification condition generator

Verification conditions

Automatic provers  
(Alt-Ergo, Gappa, CVC3, etc.)

change interpretation of FP computations

## Theorem

Let  $\varepsilon' = 2051 \cdot 2^{-60}$  and  $\varepsilon = 2050 \times 2^{-64}$ . Let  $\eta' = 2049 \times 2^{-1082}$  and  $\eta = 2049 \times 2^{-1086}$ . If we define each operation result as any real such that

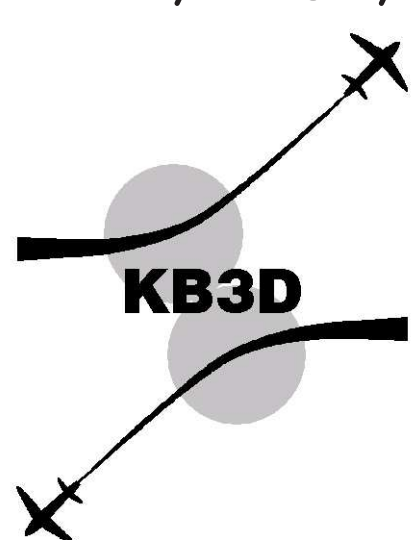
$$\begin{aligned} |x \oplus y - (x + y)| &\leq \varepsilon' \cdot (|x| + |y|) + \eta' \\ |x \ominus y - (x - y)| &\leq \varepsilon' \cdot (|x| + |y|) + \eta' \\ |x \otimes y - (x * y)| &\leq \varepsilon \cdot |x * y| + \eta \\ |x \oslash y - (x / y)| &\leq \varepsilon \cdot |x / y| + \eta \end{aligned}$$

and if we are able to deduce a property (such as a rounding error), then this property holds whatever the architecture and the compiler optimizations among commutativity, addition/subtraction associativity (for less than 16 additions), use of FMA, use of extended registers, expression factorization and unfactorization.

```
File Edit Options Buffers Tools C Help
/*@ logic integer l_sign(real x) = (x >= 0.0) ? 1 :-1; */
/*@ requires e1 <= x <= e2;
  @ ensures \abs(\result) <= 1 &&
  @ (\result != 0 ==> \result == l_sign(\exact(x)));
  @ */
int sign(double x, double e1, double e2) {
  if (x > e2) return 1;
  if (x < e1) return -1;
  return 0;
}

/*@ requires
  @ sx == \exact(sx) && sy == \exact(sy) &&
  @ vx == \exact(vx) && vy == \exact(vy) &&
  @ \abs(sx) <= 100.0 && \abs(sy) <= 100.0 &&
  @ \abs(vx) <= 1.0 && \abs(vy) <= 1.0;
  @ ensures
  @ \result != 0
  @ ==> \result == l_sign(\exact(sx)*\exact(vx) + \exact(sy)*\exact(vy))
  @ * l_sign(\exact(sx)*\exact(vy) - \exact(sy)*\exact(vx));
  @ */
int eps_line(double sx, double sy, double vx, double vy) {
  int s1 = sign(sx*vx+sy*vy, -0x1.aap-42, 0x1.aap-42);
  int s2 = sign(sx*vy-sy*vx, -0x1.aap-42, 0x1.aap-42);
  return s1*s2;
}
```

Case study  
Part of KB3D NASA  
<http://research.nianet.org/fm-at-nia/KB3D/>



Proof obligations	Alt-Ergo 0.91	CVC3 2.2 (SS)	Gappa 0.13.0	Statist
Function eps_line	✓	✓	✓	1/1
Default behavior	✓	✓	✓	
1. precondition	✓	✓	✓	
Function eps_line	✓	✓	✓	13/13
Safety	✓	✓	✓	
1. check FP overflow	✓	✓	✓	
2. check FP overflow	✓	✓	✓	
3. check FP overflow	✓	✓	✓	
4. check FP overflow	✓	✓	✓	
5. check FP overflow	✓	✓	✓	
6. precondition for user	✓	✓	✓	
7. precondition for user	✓	✓	✓	
8. check FP overflow	✓	✓	✓	
9. check FP overflow	✓	✓	✓	
10. check FP overflow	✓	✓	✓	
11. check FP overflow	✓	✓	✓	
12. precondition for use	✓	✓	✓	
13. precondition for use	✓	✓	✓	
Function sign	✓	✓	✓	6/6
Default behavior	✓	✓	✓	
1. precondition	✓	✓	✓	
2. precondition	✓	✓	✓	
3. precondition	✓	✓	✓	
4. precondition	✓	✓	✓	
5. precondition	✓	✓	✓	
6. precondition	✓	✓	✓	

## Future work

- Look into multiplication/division reordering
- Look into assembly
- know the order of the operations
- know the precision used for each operation
- know if the architecture supports FMA or not